COMPUTACIÓN EN LA NUBE

Tarea Práctica 7

Universidad de La Laguna

INDICE

1.- Lectura Cloud Native.

Como sabemos en el mundo de la informática todo va evolucionando a medida que pasa el tiempo, buscando mejorar lo que ya teníamos mejorado. Y como sabemos, el campo de las aplicaciones no iba a ser menos. Por ello, durante los últimos años se ha ido incorporado un nuevo concepto a este campo. Este es el denominado “Cloud Native Applications” o en español “Aplicaciones Nativas en la Nube”.

El término “Aplicaciones Nativas en la Nube” se basa en crear y diseñar aplicaciones que aprovechan la infraestructura en la nube bajo las siguientes características:

- Permitiendo diseñar y desplegar aplicaciones tanto en nubes públicas como privadas e híbridas.
- Permite diseñar y actualizar aplicaciones con rapidez.
- Permite mejorar la calidad y reducir riesgos.
- Son aplicaciones escalables.
- Son aplicaciones con tolerancia a fallos.
- Alta capacidad de respuesta.

En la actualidad, existen diversas tecnologías y herramientas para que permiten implementar esto. No obstante, la base consta de la unión de las siguientes tecnologías:

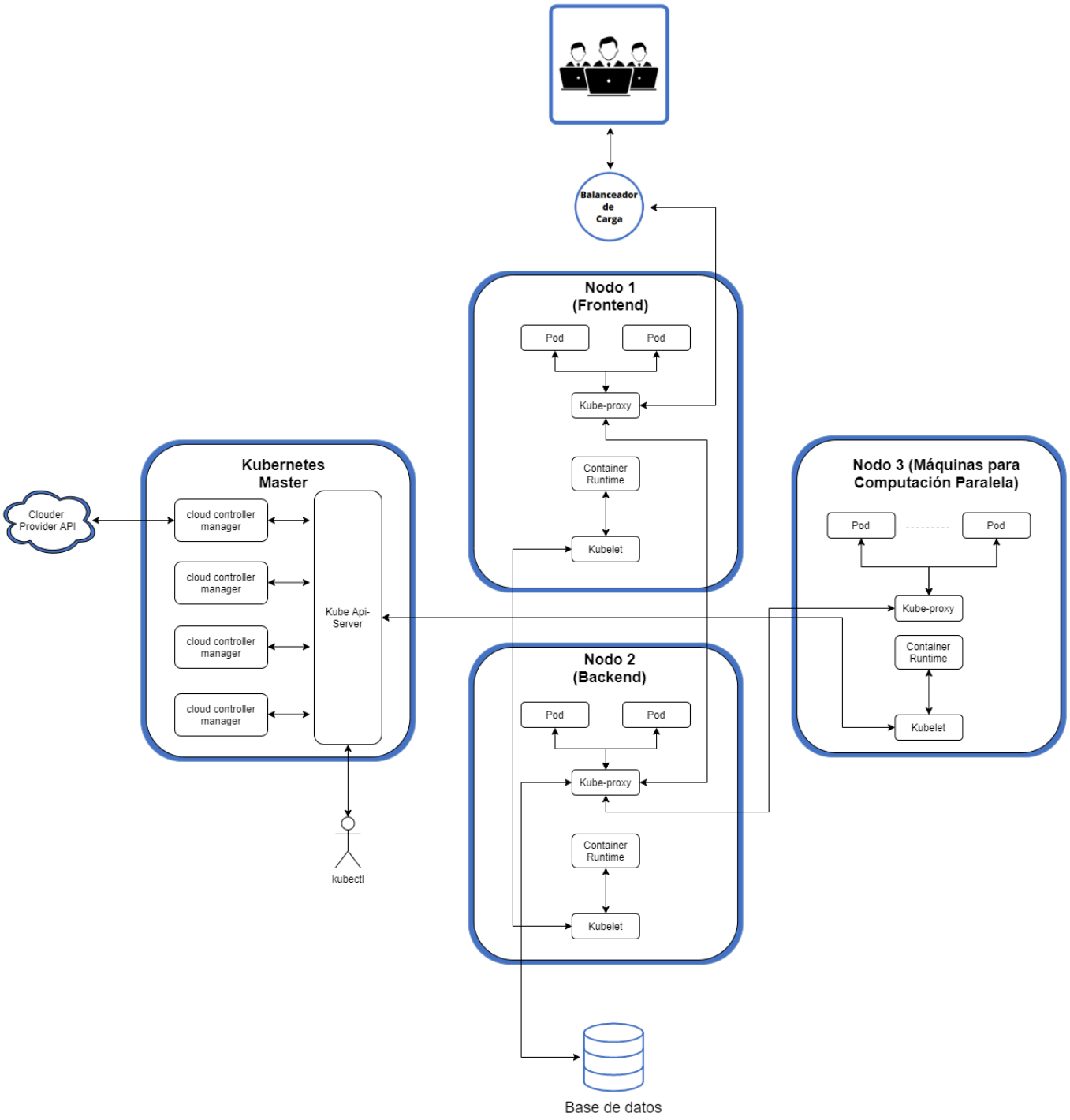
- **Crear microservicios:** servicios pequeños y de bajo acoplamiento.
- **Contenedores:** ejecutar las aplicaciones en diferentes máquinas para aprovechar al máximo las ventajas que nos da la computación en la nube.
- **Entrega continua:** permite lograr confidencialidad y velocidad a la hora de crear aplicaciones. Esto es posible ya que, no tenemos que centrarnos en los pasos posteriores al crear o modificar una aplicación (despliegue, ejecutar test,…), sino que, esto se realiza de forma automática (previamente configurado).

A modo de conclusión y, como hemos visto anteriormente, las aplicaciones nativas en la nube intentan resolver las problemáticas de las antiguas arquitecturas tradicionales, proporcionando automatización de gran parte de las acciones, tolerancia a fallos y gran escalabilidad. Este término está tan extendido que las grandes empresas tecnológicas (Google, Microsoft, Amazon, ..) ofertan está tecnología en la nube llevándola a un nivel superior. Permitiendo crear y desplegar aplicaciones sin necesidad de tener las máquinas y, facilitando su uso a precios razonables. Tanto es así, que ya existe una fundación apoyada por estas grandes empresas (Google, IBM, Intel, Cisco,.) denomina “CNCF” que se dedica a que la computación nativa en la nube sea universal y sostenible.

2.- Diseño aplicación Cloud Native.

2.1.- Explicación y Diseño creado.

Como hemos visto en el documento proporcionado por el profesor ("Chapter 3. Designing Cloud Native Applications"), existen diversas formas de diseñar y desplegar una aplicación Cloud Native. En mi caso, he optado por usar “Kubernetes“ una tecnología que nos permite implementar y administrar contenedores a gran escala de forma sencilla y automatizada. Por ello, el diseño propuesto es:



Como podemos ver en la imagen anterior, es un diseño basado en Kubernetes donde podemos ver el flujo de trabajo de nuestro framework implementado durante el curso. En los puntos posteriores, podremos ver una explicación de cada sección del diseño por separado.

Gracias a este diseño podremos resolver todos los fundamentos planteados en una aplicación nativa en la nube (vistas en el documento aportado por el profesor). Estas las podremos ver a continuación:

- **Excelencia operacional:** se basa en poder construir, medir, aprender y mejorar. Gracias a Kubernetes este concepto es sencillo de realizar. Este se subdivide en:
  - Automatiza todo.
  - Monitorear todo.
  - Documenta todo
  - Hacer cambios incrementales
  - Diseño para el fracaso
- **Seguridad:** de igual forma, nuestra estructura consta con un balanceador de carga que junto con la estructura de Kubernetes y de la nube seleccionada impedirán el acceso a peticiones maliciosas. Esta estructura sigue con el concepto de “defensa en profundidad” ya que, cada estructura dispone de su propia seguridad.
- **Fiabilidad y disponibilidad:** gracias a Kubernetes este término está garantizado. Esto es debido a que Kubernetes es tolerante a fallos, es decir, en caso de que una instancia (pod) se caiga puede eliminarla y levantar otra prácticamente al instante (y de forma automática).
- **Escalabilidad y costo:** gracias a Kubernetes la escalabilidad está garantizada hasta un punto (cuando no se puedan crear más máquinas debido a la tarifa contratada). Sin embargo, este punto lo tocaremos en el apartado 2.3.

2.2 Explicación secciones del diseño

2.2.1 Balanceador de carga.

El balanceador de carga es un dispositivo que nos permitirá poder distribuir las peticiones de un nodo a otro con facilidad. Aunque al principio no tiene mucha utilidad, ya que, podríamos conectar la llamada del cliente directamente al proxy del nodo, está pensado para poder ampliar fácilmente nuestro modelo de cara al futuro y, añadir otro nodo de Frontend y solo tener que añadir la nueva ip al lugar correspondiente.

Este dispositivo es una máquina que (en un principio) no está configurada con Kubernetes. Existen muchas formas de implementar esta máquina, en mi caso se usaría la tecnología denominada “Nginx”, ya que, es una tecnología ampliamente extendida y, porque ya la he usado con anterioridad.

2.2.2 Kubernetes Master.

Es la máquina principal para la implantación de Kubernetes. Esta máquina nos permitirá realizar todas las acciones necesarias para que el estado del clúster coincida con el deseado. Por ello, esta máquina realizará diferentes tareas de forma automática (parar o arrancar contenedores,…). De igual forma, nos permitirá la posibilidad de escalar el número de réplicas de una aplicación dada (de forma automática siempre que sea necesario).

Esta máquina dispone de diversos elementos y no podremos entrar en detalle en todos estos. No obstante, el elemento denominado “Cloud Controller Manager” es un elemento muy importante. Siguiendo con la lectura de la práctica anterior (decir cuánto cuesta implantar una esquema sencillo en una nube) esta máquina es la que se vinculará con una de estas nubes (la elegida previamente) para realizar las acciones necesarias (autentificación, crear máquinas,..).

2.2.3 Nodo 1 (Frontend).

Esta instancia denominada “Nodo 1 (Frontend)” es una máquina de trabajo en Kubernetes (puede ser máquina virtual o física) y es la que nos va a permitir realizar las siguientes acciones:

- Crear nuevos pods (contenedor) con el código del Frontend previamente aportado.
- Conectarnos con un pod (contenedor) con el código de nuestro Backend y, realizar las accione pertinentes.
- Conectarnos con la máquina Kubernetes master y, si es necesario crear nuevos pods (aumenta la carga).

Como vemos, esta máquina realizará todas estas acciones de forma automática sin prácticamente nosotros hacer nada (tras configurarlo todo).

2.2.4 Nodo 2 (Backend).

La instancia denominada “Nodo 2 (Backend)” es una máquina de trabajo en Kubernetes y, nos va a permitir realizar las siguientes acciones de forma automatizada:

- Crear nuevos pods (contenedor) con el código del Backend previamente aportado.
- Conectarnos con uno o varios pods (contenedores) para ejecutar las acciones necesarias (código paralelo).
- Conectarnos con la máquina Kubernetes master y, si es necesario crear nuevos pods (aumenta la carga).
- Conectarse (cuando sea necesario) a la base de datos.

2.2.5 Nodo 3 (Máquinas para Computación en la Nube).

La instancia denominada “Nodo 3 (Máquinas para Computación en la Nube)” es una máquina de trabajo en Kubernetes (puede ser máquina virtual o física) y es la que nos va a permitir realizar las siguientes acciones de forma automatizada:

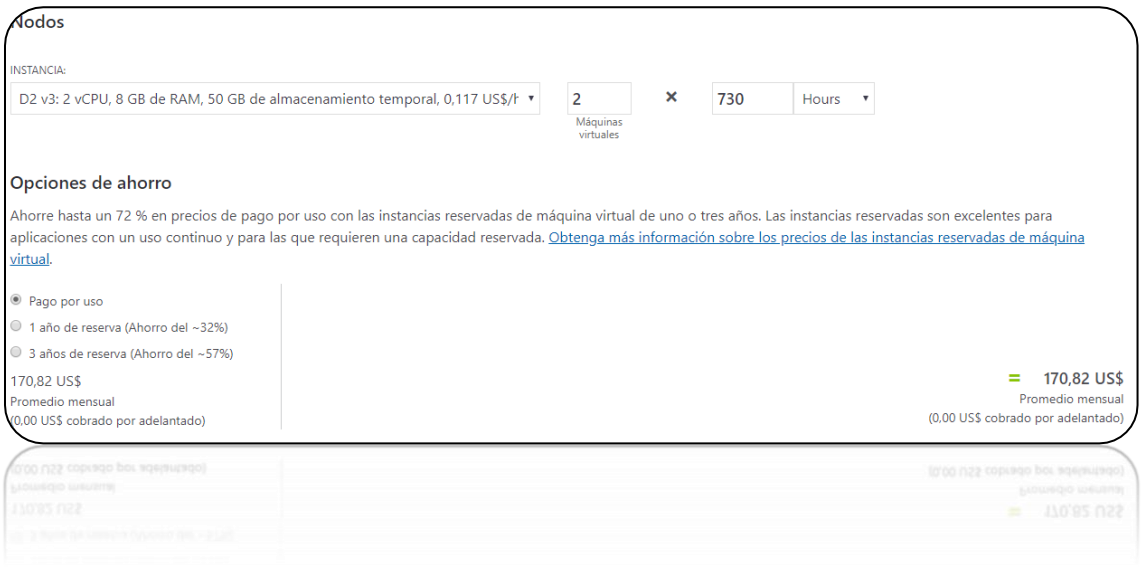
- Crear nuevos pods (contenedor) que nos permitirá realizar más acciones paralelas.
- Devolver el resultado de cada acción paralela.
- Conectarnos con la máquina Kubernetes master y, si es necesario crear nuevos pods (aumenta la carga).

2.3 Coste estimado en Microsoft Azure.

Aprovechando el ejercicio de la práctica anterior (análisis de costes de despliegue) se ha realizado un análisis para el despliegue de esta estructura en la nube que se me ha concedido para el ejercicio (Microsoft Azure). Por ello, se han realizado dos presupuestos uno básico (únicamente se alquila dos máquinas) y, uno más avanzadas (alquilando 4 máquinas). Esto es debido a que inicialmente y, hasta que la aplicación sea rentable se utilizaría el primer presupuesto. No obstante, si todo nos sale favorablemente deberemos de aumentar las máquinas para abastecer a una demanda superior. Estos presupuestos los podremos ver en los puntos que vienen a continuación.

2.3.1 Presupuesto simple

Este presupuesto consta de dos máquinas con 2 CPU con 8GV de RAM y 50gb de almacenamiento. Este dispone de un presupuesto de 170,83 $ (sin permanencia), 115,84 $ (permanencia de un año) y, 74,11 $ (permanencia de 3 años) al mes. De igual forma, a continuación, podremos ver una muestra de ello:



2.3.2 Presupuesto avanzado.

Este presupuesto consta de cuatro máquinas con 2 CPU con 8GV de RAM y 50gb de almacenamiento. Este dispone de un presupuesto de 341,64 $ (sin permanencia), 231,67 $ (permanencia de un año) y, 148,22 $ (permanencia de 3 años) al mes. De igual forma, a continuación, podremos ver una muestra de ello:

3. Apache Spark.

3.1 Instalación.

Para la instalación de Apache Spark se ha seguido el tutorial que podremos encontrar en el link "https://computingforgeeks.com/how-to- install-apache-spark-on-ubuntu-debian/". En este podremos encontrar los siguientes puntos:

- Instalar Java mediante el comando "sudo apt install default-jdk".
- Descargar Apache Spark desde la página oficial (https://spark.apache.org/downloads.html).
- Descomprir lo descargado.
- Mover a la ruta "/opt/spark" mediante el comando "sudo mv spark-2.4.2-bin-hadoop2.7/ /opt/spark".
- Abrimos el archivo de configuración de bash y añadimos las siguientes líneas:

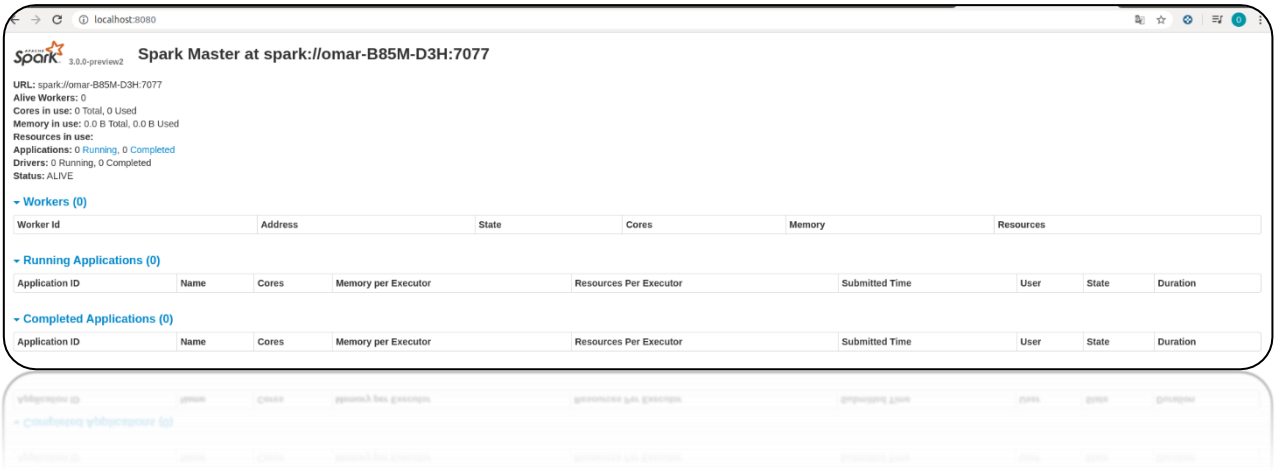
```
export SPARK_HOME=/opt/spark
export PATH=$PATH:$SPARK_HOME/bin:$SPARK_HOME/sbin
```

- Activamos los cambios mediante el comando "source ~/.bashrc".

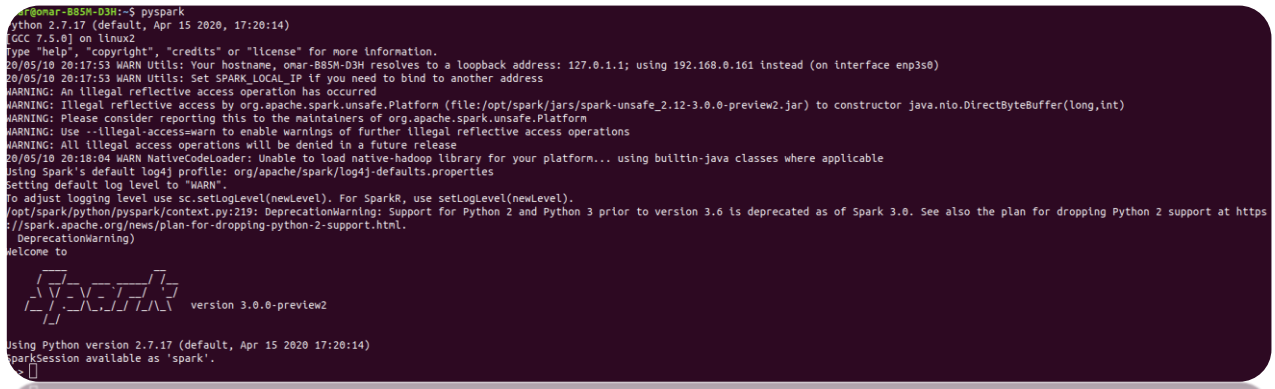
- Iniciamos un servidor maestro mediante los siguientes comandos:
    - start-master.sh
    - ss -tunelp | grep 8080
- Iniciamos el proceso de Spark Worker mediante los comandos:
    - start-slave.sh spark://ubuntu:7077
    - locate start-slave.sh

Por último, comprobamos que todo se ha instalado correctamente mediante las siguientes pruebas:

Accedemos a la url “http://localhost:8080/”



Abrimos la terminal de pyspark



3.2 Prueba ejemplo.

3.2.1 Usando Python.

Para este apartado, he usado uno de los ejemplos que nos propone la propia página de Spark (https://spark.apache.org/examples.html). Este ejemplo nos permite estimar π "lanzando dardos" en un círculo. Seleccionamos puntos aleatorios en el cuadrado de la unidad ((0, 0) a (1,1)) y vemos cuántos caen en el círculo unitario. La fracción debe ser π / 4, por lo que usamos esto para obtener nuestra estimación. El código y la ejecución de este código la podremos a continuación:

Código

```
import findspark
findspark.init("/opt/spark")
import random
from pyspark import SparkContext
from time import time


start_time = time()
sc = SparkContext(appName="EstimatePi")
def inside(p):
    x, y = random.random(), random.random()
    return x*x + y*y < 1
NUM_SAMPLES = 1000000
count = sc.parallelize(range(0, NUM_SAMPLES)) \
             .filter(inside).count()
print("Pi is roughly %f" % (4.0 * count / NUM_SAMPLES))
elapsed_time = time() - start_time
print("Elapsed time: %0.10f seconds." % elapsed_time)
sc.stop()
```

Ejecución

```
omar@omar-B85M-D3H:~/Escritorio/Pruebas Apache Spark$ python3 Prueba1.py
20/05/10 21:41:10 WARN Utils: Your hostname, omar-B85M-D3H resolves to a loopback address: 127.0.1.1; using 192.
168.0.161 instead (on interface enp3s0)
20/05/10 21:41:10 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by org.apache.spark.unsafe.Platform (file:/opt/spark/jars/spark-unsafe_2.12-3
.0.0-preview2.jar) to constructor java.nio.DirectByteBuffer(long,int)
WARNING: Please consider reporting this to the maintainers of org.apache.spark.unsafe.Platform
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release
20/05/10 21:41:21 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin
-java classes where applicable
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Pi is roughly 3.141344
Elapsed time: 13.9817631245 seconds.
```

3.2.2 Usando la terminal de pyspark.

Para este ejercicio, he usado un tutorial que podremos encontrar en la página "http://exponentis.es/ejemplo-de-uso-de-pyspark-en-linux-y-algunos-comandos-basicos-de-transformacion-accion-en-spark". En este nos explican algunos comandos que podremos usar en la consola de pyspark. Para ello, he realizado los siguientes pasos:

- Copiamos uno de los libros de la página de "https://www.gutenberg.org". Posteriormente, ejecutamos el comando "curl https://www.gutenberg.org/files/50133/ 50133-8.txt > dunwich.txt".
- Cargamos el fichero dentro de una variable mediante el comando "miRDD = sc.textFile("file:///home/omar/libros")".

Tras hacer estos pasos se han probado diferentes comandos de pyspark. Algunos ejemplos los podremos ver a continuación.

Número de líneas

Para ello, usaremos la función "count()". Esto lo podremos ver en la siguiente imagen:

```
>>> miRDD.count()
2117
```

Primera línea

Para ello, usaremos la función "first()". Una prueba de ello la podremos ver a continuación:

```
>>> miRDD.first()
u'The Project Gutenberg EBook of The Dunwich Horror, by H. P. Lovecraft'
```

Número de particiones usadas

Para ello, usaremos la función "getNumPartitions()". Esto lo podremos ver a continuación:

```
>>> miRDD.getNumPartitions()
2
```

Ejemplo líneas del fichero

Esta funcionalidad nos permite coger líneas aleatorias dentro del fichero. Esto lo podremos ver a continuación:

```
>>> miRDD.takeSample(False, 5)
[u'half-articulate thunder-croakings drawn? Presently they began to gather', u'p
oor Curtis Whateley began to knit back into a sort of continuity; so', u'to resc
ue the fallen telescope and wipe it clean of mud. Curtis was', u'May Eve of 1915
 there were tremors which even the Aylesbury people', u'Lake City, UT 84116, (80
1) 596-1887. Email contact links and up to']
```